架构

2020年8月20日 20:29

-
- 架构

1. 单体架构
    a. 特点
        i. 一个归档包包含了应用所有功能的应用程序， 我们通常称之为单体应用
        ii. 架构单体应用的架构风格， 我们称之为单体架构
        iii. 是一种比较传统的架构风格
    b. 缺点
        i. 复杂性逐渐变高
        ii. 技术债务逐渐上升
        iii. 部署速度逐渐变慢
        iv. 阻碍技术创新
        v. 无法按需伸缩
    c. 演化
        i. 简化增删改等操作的辅助工具（ORM等）
        ii. 水平分层开发和维护（MVC等）
        iii. 应用拆成互不相干的几个应用，以提升效率（垂直分层开发）
        iv. 核心心业务抽取出来，作为独立的服务（分布式服务框架(RPC)）
        v. 服务越来越多，兼容性、扩展性、服务治理等问题逐渐涌现（采用SOA）
2. SOA面向服务的体系结构（service-oriented architecture）
    a. 构造分布式计算的应用程序的一种架构模式思想
        i. 将应用程序功能作为服务发送给最终用户或者其他服务
        ii. 是一种粗粒度、松耦合服务架构，服务之间通过简单、精确定义接口进行通讯，不涉及底层编程接口和通讯模型
    b. 服务：在业务系统中被发布出来供用户使用，能够完成一个完整业务过程的功能
        i. 服务着眼于完整的业务
        ii. 服务的粒度虽然相对粗放，但却可控，目标是重用
        iii. 集成的目的是形成一个新的服务
        iv. 需屏蔽细节
        v. 让各业务系统保持松散
    c. 特点：自治（理）性、粗粒度、可见性、无状态、幂等性、可重用性、可组合
    d. 优点
        i. 面向服务的体系结构，是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来
        ii. 把模块拆分，使用接口通信，降低模块之间的耦合度
        iii. 把项目拆分成若干个子项目，不同的团队负责不同的子项目

iv. 增加功能时只需要增加一个子项目，调用其它系统的接口就可以
v. 可以灵活的进行分布式部署

e. 缺点
i. 系统之间交互需要使用远程通信，接口开发增加工作量
ii. 业务功能的增多SOA的服务会变得越来越复杂，本质上看没有因为使用SOA而变的更好
iii. 新技术的发展导致SOA的开发方式和步骤也不断地演化

f. 演化
i. 1.0时代：基于SOAP的跨企业的Web service交互
ii. 2.0时代：支持企业内部系统间轻量级访问，支持服务治理
iii. 3.0时代：服务进一步分层和微服务化

g. 实现
i. 是一种复杂的架构风格和理念，并非是一种技术或者产品
ii. 其服务调用接口的实现技术繁多，如WebService、WCF、Hessian、RMI等
iii. 实践SOA不仅需要解决服务调用的问题，还包括服务编排、服务治理、服务路由、服务监控等
iv. 在大型分布式系统中，SOA被广泛实践，但是在不同的应用场景中，设计方法也大不相同

h. 角色
i. 服务提供者：发布自己的服务，并且对服务请求进行响应
ii. 服务注册中心：注册已经发布的web service，对其进行分类，并提供搜索服务
iii. 服务请求者：利用服务中心查找所需要的服务，然后使用该服务

i. 操作
i. 发布：为了使服务可访问，发布服务描述以使服务使用者可以发现它
ii. 查找：服务请求者定位服务，方法是查询服务注册中心来找到满足其标准的服务
iii. 绑定：在检索到服务描述之后，服务使用者继续根据服务描述中的信息来调用服务

3. WebService：一种经典的，常见的SOA服务接口实现技术之一
a. Web service是技术规范，SOA是设计原则。从本质上讲，SOA是一种架构模式，而web service是利用一组标准实现的服务
b. 特点
i. 良好的封装性：Web Services是一种部署在Web上的对象，因此具有对象的特点，即良好的封装性。这样服务使用者只能看到对象提供的通用接口和功能列表，而不用关心服务的实现细节
ii. 松耦合：只要Web Services的调用接口不变，其内部变更对调用者来说是透明的。使用标准协议规范Web服务是基于XML的消息交换机制，其所有公共的协约都采用Internet开放标准协议进行描述、传输和交换。相比一般对象而言，其调用更加规范化，便于机器理解
iii. 高度可集成性：由于Web Services采取简单的、易于理解的标准协议作为组件描述，因此完全屏蔽了不同软件、平台的差异，使它们之间可以通过Web Services来实现互操作
iv. 易于构建：要构建Web Services，开发人员可以使用任何常用的编写语言（如Java、C#、C/C++或Perl等）及其现有的应用程序组件
c. 应用场合
i. 跨防火墙的通信：把应用程序的中间层换成Web Services，可以从客户端直接调用服务，而不需要建立

额外的ASP页面。同时作为中间层的Web Services完全可以在应用程序集成场合下重用。这样不仅减少了开发周期和代码复杂度，还能够增强应用程序的重用性和可维护性

ii. 应用程序集成：通过Web Services把应用之间需要“暴露”给对方的功能和数据，以服务接口的形式提供给调用者，而不用去在意各应用程序之间异构性，因为Web Services是采用开放的、标准的、跨平台的协约来执行的。

iii. B2B的集成：Web Services是实现B2B应用程序高效集成的重要技术之一，电子商务公司可以根据业务需求将应用的接口“暴露”给指定的客户或供应商，以方便客户和供应商高效地完成电子业务。

iv. 软件和数据重用：组件提供商可以把组件变成Web Services，并把相应的服务接口提供给服务使用者。这样，服务使用者无需将服务组件下载到本地并安装，而是直接在应用中调用服务，获取所需的数据

d. 工作原理

i. 服务提供者注册和发布自己的服务在服务注册中心中，并对服务请求进行响应

ii. 服务注册中心担任中介的作用，一边接收服务提供者发来的服务，一边供服务提供者在其统一目录中查找合适的服务

iii. 服务使用者是根据具体的应用需求调用服务的

iv. 在典型情况下，服务提供者托管可通过网络访问特定的软件模块，定义Web Services的服务描述并将服务发布到服务注册中心统一目录中；服务请求者使用查找操作从注册中心中检索特定的服务，然后使用服务描述与服务提供者进行绑定并调用相应的服务

e. 实现方案

i. SOAP简单对象访问协议（Simple Object Access Protocol）

1) 一种标准化的通讯规范，主要用于Web服务（web service）中

2) 用一个简单的例子来说明 SOAP 使用过程，一个 SOAP 消息可以发送到一个具有 Web Service 功能的 Web 站点，例如，一个含有房价信息的数据库，消息的参数中标明这是一个查询消息，此站点将返回一个 XML 格式的信息，其中包含了查询结果（价格，位置，特点，或者其他信息）。由于数据是用一种标准化的可分析的结构来传递的，所以可以直接被第三方站点所利用

ii. REST表述性状态转移（Representational State Transfer）

1) 采用Web 服务使用标准的 HTTP 方法 (GET/PUT/POST/DELETE) 将所有 Web 系统的服务抽象为资源，REST从资源的角度来观察整个网络，分布在各处的资源由URI确定，而客户端的应用通过URI来获取资源的表述（representation）。REST是一种软件架构风格而非协议也非规范，是一种针对网络应用的开发方式，可以降低开发的复杂性，提高系统的可伸缩性

iii. XML-RPC远程过程调用（Remote Procedure Call，RPC)

1) 通过XML将调用函数封装，并使用HTTP协议作为传送机制。后来在新的功能不断被引入下，这个标准慢慢演变成为今日的SOAP协定。XML-RPC协定是已登记的专利项目。XML-RPC透过向装置了这个协定的服务器发出HTTP请求。发出请求的用户端一般都是需要向远端系统要求呼叫的软件

iv. XML-RPC已慢慢的被SOAP所取代，现在很少采用了，但它还是有版权的；SOAP在成熟度上优于REST；在效率和易用性上，REST更胜一筹；在安全性上，SOAP安全性高于REST，因为REST更关注的是效率和性能问题

v. SOAP是Web Services的核心技术，且现在大多数Web Services都是基于SOAP的，其应用开发也相对成熟

vi. REST是一项新技术，它不是协议、不是架构，严格来说也不属于Web Services；它只是一种抽象的软件设计观念，要求从资源的角度来考虑问题。Web Services的核心架构其实就是**SOAP技术**，而**REST是理念**

4. SOAP：(Simple Object Access Protocol，简单对象访问协议）是一种基于XML的、

轻量级的、跨平台的数据交换协议。SOAP不仅描述了数据类型的消息格式及一整套串行化规则，包括结构化类型和数组，还描述了如何使用HTTP来传输消息。SOAP提供了应用程序之间的交互能力

a. 包括
    i. SOAP Envelope：用于定义一个描述消息中的内容、发送者、接收者、处理者及如何处理的整体表示框架，基于XML
    ii. SOAP编码规则：定义了一套编码机制用于交换应用程序定义的数据类型的实例
    iii. SOAP RPC：表示远程过程调用和应答的协定
    iv. SOAP绑定：定义了一种使用底层传输协议来完成节点间交换SOAP消息的约定
b. 协议栈
    i. 传输网络：HTTP、SMTP等
    ii. SOAP消息：SOAP
    iii. 服务描述：WSDL
    iv. 服务发现、发布：UDDI
    v. 服务流程：WSFL
c. 模式
d. WSDL（Web Services Description Language，Web服务描述语言）是一种基于XML的、专门用于描述Web Services的语言。通过WSDL可以对服务的功能信息、功能参数的消息类型、协议绑定信息和特定服务的地址信息进行描述
e. UDDI（Universal Description、Discovery and Integeration)统一描述、发现和集成
f. 实现框架
    i. CXF-apche
    ii. Jwx，也就是jax-ws--sun
    iii. Axis2-axis的升级轻量级版本

5. RESTful API服务接口
    a. REST原则
        i. 前端设备层出不穷（手机、平板、桌面电脑、其他专用设备……），促使前后端分离，方便不同的前端设备与后端进行通信，导致API架构的流行
        ii. RESTful架构，因结构清晰、符合标准、易于理解、扩展方便，是目前流行的一种互联网软件架构
        iii. RESTful API是目前比较成熟的一套互联网应用程序的API设计理论
        iv. 由（HTTP早期主要设计者、Apache基金会首任主席）Roy Thomas Fielding在他2000年的博士论文中提出：在符合架构原理的前提下，理解和评估以网络为基础的应用软件的架构设计，得到一个功能强、性能好、适宜通信的架构。他对互联网软件的架构原则，定名为REST
        v. REST= Representation + State + Transfer 表现层状态转化
    b. RESTful架构：符合REST原则的架构
        i. 每一个URI代表一种资源
        ii. 客户端和服务器之间，传递这种资源的某种表现层
        iii. 客户端通过HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"

c. URI设计规则
    i. 资源表示一种实体，URI使用**名词**表示，不应该包含动词；一般来说，数据库中的表都是同种记录的"集合"（collection），所以URI中的名词应该使用**复数**，例：GET /zoos/ID：获取某个指定动物园的信息
    ii. 如果某些动作是HTTP动词表示不了的，应该把动作做成一种资源，例：POST /transaction/from=1&to=2&amount=500.00
    iii. 参数的设计允许存在冗余，即允许API路径和URL参数偶尔有重复
    iv. 常见参数：
        1) ?limit=10：指定返回记录的数量
        2) ?offset=10：指定返回记录的开始位置
        3) ?page=2&per_page=10：指定第几页，以及每页的记录数
        4) ?sortby=name&order=asc：指定返回结果按照哪个属性排序，以及排序顺序
d. HTTP动词
    i. GET：从服务器取出资源（一项或多项）
    ii. POST：在服务器新建一个资源
    iii. PUT：在服务器更新资源（客户端提供改变后的完整资源）
    iv. PATCH：在服务器更新资源（客户端提供改变的属性）
    v. DELETE：从服务器删除资源
    vi. HEAD：用于获取某个资源的元数据（metadata）
    vii. OPTIONS：用于获取某个资源所支持的Request类型
e. 对比
    i. SOAP：技术成熟稳定、安全性高、银行，证券等大型企业级解决
    ii. REST：轻量级解决方案，性能较优、物联网公司、普通公司业务集成、移动终端

6. ESB（Enterprise Service Bus）全名：企业服务总线，是SOA架构思想的一种实现思路
a. 问题：SOA企业级应用中多个业务系统，甚至各业务系统负责的功能职责还存在重叠。这些系统采用不同时代的编程语言、编程框架、通讯协议、消息格式和存储方案。需要一个成熟稳定、兼容易用的中间层进行协调
b. 目标：服务的集成和管理
c. 作用
    i. 多调用协议支撑和转换
    ii. 多消息格式支撑和转换
    iii. 服务监控管理（注册、安全、版本、优先级）
    iv. 服务集成和编排
    v. 服务的版本控制
d. EAI到ESB
    i. ESB是EAI的进化
    ii. ESB企业服务总线技术是在SOA架构之后出现的，在这之前为了集成多个系统而使用最多的技术思路是EAI（Enterprise Application Integration）：

企业应用集成

iii. EAI技术并没有一个统一的标准，而是对不同企业集成业务系统手段的统一称呼

e. 实现

i. 商用产品：IBM WebSphere ESB和IBM WebSphere Message Broker（WMB）、Oracle Service Bus (OSB)

ii. 开源产品：Mule ESB、Apache ServiceMix

7. 微服务

a. In short, a microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, such as HTTP API. These services are built around business capabilities and are independently deployable by fully automated deployment machinery. There is minimal centralized management of these services, which may be written in different programming languages and use different data storage technologies. -----Martin Fewler

i. a suite of small services：微型服务的集合

ii. running in its own process：每个服务运行自己的进程

iii. communicating with lightweight mechanisms：服务之间通过轻量级通信机制如HTTP进行通信

iv. built around business capabilities：服务是基于业务功能划分进行设计

v. independently deployable：每个服务都可以独立部署

vi. minimal centralized management of these services：服务之间的集中管理程度很低

b. 应用场景

i. 一开始你有一个很成功的关键业务应用，后来就变成了一个巨大的，无法理解的怪物。因为采用过时的，效率低的技术，使得雇佣有潜力的开发者很困难。应用无法扩展，可靠性很低，最终，敏捷性开发和部署变的无法完成

ii. 解决思路：不是开发一个巨大的复杂的应用，而是将应用分解为小的、互相连接的微服务。逐一分别开发

c. 微服务 VS SOA

i. 粒度不同：细 vs 粗

ii. 服务的集成： 相对松散 vs ESB

d. 实现

i. Docker：一种开源的应用容器引擎，轻量级的虚拟机

1) 服务应用的宿主

2) 一致的运行环境

3) 轻松的迁移

4) 持续交付和部署

5) 维护和扩展

6) 比传统虚拟机更高效、快速、方便

7)

特性	容器	虚拟机
启动速度	秒级	分钟级
性能	接近原生	较弱
内存代价	很小	较多
硬盘使用	一般为MB	一般为GB
运行密度	单机支持上千个容器	一般几十个
隔离性	安全隔离	完全隔离
迁移性	优秀	一般

ii. Spring Cloud：spring生态圈中的一系列框架的集成，提供了一个易部署和易维护的分布式系统开发工具包
   1) 基于SpringBoot
   2) 为小型企业和公司提供了一套标准化的、全站式的分布式开发解决方案